Impact of used programming language for K-12 students' understanding of the loop concept

Monika Mladenović* and
Saša Mladenović

Faculty of Science,
University of Split,
Ruđera Boškovića 33, 21000 Split, Croatia
Email: monika.mladenovic@pmfst.hr
Email: sasa.mladenovic@pmfst.hr
*Corresponding author

Žana Žanko

Elementary School "Mejaši",
Mejaši 20, 21 000 Split, Croatia
Email: zana.zanko@skole.hr

Abstract: Block-based programming languages are becoming a favourite learning tool for programming novices while the traditional way of teaching programming mostly uses text-based programming languages. The purpose of this study was to compare the impact of used visual and textual programming languages on K-12 students' understanding of the loop concept. Participants were 312 elementary school students from 5th to 8th grade using visual programming language Scratch (n = 59), and textual programming languages Logo (n = 185) and Python (n = 68). Tests for all languages were equivalent, differing only in the used programming language. Results showed that students achieved statistically significant higher scores when using block-based programming language compared to students using textual programming languages. These results show that K-12 students need concrete experience to understand abstract concepts, as the loop concept, which Scratch as a block-based programming language provides.

Keywords: programming; loop; elementary school; Scratch; text-based programming languages.

Reference to this paper should be made as follows: Mladenović, M., Mladenović, S. and Žanko, Ž. (2020) 'Impact of used programming language for K-12 students' understanding of the loop concept', *Int. J. Technology Enhanced Learning*, Vol. 12, No. 1, pp.79–98.

Biographical notes: Monika Mladenović is a teaching assistant at the Faculty of Science, University of Split, Croatia, at the Computer Science Department since 2017. From 2003 to 2012 she was working as a programmer and is still periodically working as a programmer as an external associate. From 2012 to 2017 she was working as Computer science teacher at elementary school. She is a PhD student at the Faculty of Science, University of Split (Croatia). Her research interest includes teaching computer programming and learning.

Saša Mladenović is an Associate Professor of Computer Science at the Faculty of Science, University of Split, Croatia. He teaches some doctoral, graduate

and undergraduate courses at the Department of Informatics. He received his PhD in Computer science at the Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture in Split (FESB). From 1999 to 2006 he was acting as Technical manager of the Toll collection system department of Ecsat, Croatia – the company responsible for software development of CS group Designer, Integrator, and operator of mission-critical systems, France. His research interests include problems in teaching programming, interoperability, intelligent technologies like ontology and multi-agent systems, especially engineering applications of intelligent technologies. He has authored or co-authored more than 30 scientific papers. He is IEEE member since 1996.

Žana Žanko is computer science teacher working in Elementary School Mejaši in Split, Croatia since 2001. She is a PhD student at the Faculty of Science, University of Split, Croatia. Her research interest includes misconceptions in learning computer programming.

1 Introduction

In the last quarter of the 20th century, programming language designers have been trying to develop programming languages for novice programmers. Before asking if we need to develop programming skill with all students, we should consider that until nonprogrammers can program computers themselves, they will not be able to exploit the full power of the computer (Smith et al., 2000). In the 21st century novices in programming are K-12 students as well as undergraduate students. Guided by foreseen problems programming language designers have created a new generation of programming languages intended to be used by the younger generation of programming novices. Those languages are based on the visual paradigm and are also known as visual block-based languages. Unlike traditional text-based programming languages, block-based programming languages, like Scratch, are created to be free of syntax errors. Thus, students can focus on problem-solving instead of dealing with syntax problems. Furthermore, the programming context shifts from math-based problems to programming games, storytelling, animations and such (Honey, 2014; Xinogalos et al., 2017). Thus, block-based programming languages, like Scratch, would be a good introduction for programming novices at K-12, but also at K-16 level (Dorling and White, 2015).

When teaching programming novices, there are several crucial questions. Those questions, sorted by the importance, are: What age are the novices? Which programming language to use? What programming context to choose?

All these questions are strongly related and mutually dependent. Choosing the programming language for programming novices is directly related to the students' age, for, there is a significant difference between K-12 and undergraduate level novices. Also, the programming context is consequently related to the programming language.

The goal of this study was to compare the effect of two different approaches for introductory programming lectures for K-12 novices. One strategy fits the traditional way of teaching programming with the use of text-based programming languages. The second approach fits a game-based approach context with the use of block-based programming language. We studied elementary school students (K-12 level) to determine the understanding of one of the basic programming concepts – loop, using three different programming languages. Three different programming languages – Scratch, Logo, and Python – were chosen (Dorling and White, 2015) as they fulfil informatics curriculum

requirements of the Republic of Croatia (Ministry of Science Education and Sports of the Republic of Croatia, 2005). We wanted to identify the differences in learning programming concepts based on learning and teaching settings. We compared the results obtained from the research on the population of 312 elementary school children, in the Republic of Croatia, using Scratch, Logo, and Python programming languages. In this paper, we present the results of that research.

2 Background

2.1 Computer science (CS) and programming

Since its beginnings, CS has been a unique combination of math, engineering, and science. It is not one, but all three. If the focus is on a single subset, the uniqueness of the field cannot be expressed (Denning, 2009). The ubiquitous question is whether or not all students should study computer science. Opponents claim no, which leads us to a vicious cycle due to the subjects not being part of the schooling system (Webb et al., 2016). Students form opinions and attitudes towards computing at an early age (Yardi and Bruckman, 2007) and often decide to pursue a career in the sciences by eighth grade (Tai, 2006), while a minority of students do not find science engaging until their college years (Maltese and Tai, 2010). Furthermore, it is more likely that students will leave the CS major if they have not had any exposure to CS by high school and come from schools where the workload was significantly lighter (Biggers et al., 2008).

Programming, as part of CS, is proven as a successful way of practising problem-solving and abstract thinking a.k.a., computational thinking (Papert, 1980). There is a decline in enrolment CS and programming courses at all levels, and many students have a negative perception of computing (Cassel et al., 2007; Kinnunen and Malmi, 2006; Violino, 2009; Yardi and Bruckman, 2007). Students do not see computing as an environment for creativity enhancement and believe that it focuses on tedious details instead of essential topics in problem-solving (Cassel et al., 2007; Kinnunen and Malmi, 2006; Violino, 2009; Yardi and Bruckman, 2007). Also, many computer scientists believe that novices need a mediating environment to scaffold learning into more expert programming behaviours (Dann et al., 2012; Fincher and Utting, 2010); it is not that any particular programming language is ambiguous, but the pedagogic approach is. When it comes to programming, there are some changes in those directions. In the last decade, there are new visual programming languages like Scratch, designed specifically for children who are programming novices. Many studies have shown the benefit of such an approach in motivation for, and understanding of programming concepts (Dann et al., 2012; Maloney et al., 2008a; Ben-Bassat Levy et al., 2003; Armoni et al., 2015; Malan et al., 2007; Maloney et al., 2008b; Mladenović et al., 2016a; Mladenović et al., 2016b; Lewis, 2010; Mladenović et al., 2017b).

2.2 Programming environments

Students experience programming as hard and boring (Repenning et al., 2010; Yardi and Bruckman, 2007) especially when it comes to K-12 children.

Abstraction skills are essential in programming (Kramer, 2007; McCracken et al., 2001; Moser, 1997; Sanders and Thomas, 2007) and at the same time good tool for practising abstract thinking (Dann and Cooper, 2009). Accordingly, the problems for

many students start at an early stage of learning when they need to understand and apply the abstract programming concepts (Gomes and Mendes, 2007; Lahtinen et al., 2005; McKenna, 2004).

According to Piaget's research, students need the concrete experience to understand abstraction, and computers can make the abstract concrete (Turkle and Papert, 1992). Attracting a more diverse student population and retaining them in undergraduate computing programs should involve teaching and learning strategy beginning with the concrete in a context familiar to students and then gradually leading to an understanding of the abstract concepts (Dann and Cooper, 2009). Students understand what they can see, and if they cannot see what a program is doing, they struggle to understand it. The question is: How can we make abstract programming concepts concrete?

Text-based procedural programming languages such as Python require abstract thinking (White and Sivitanides, 2009). Furthermore, particularly K-12 students, have to struggle with the programming language syntax instead of focusing on problem-solving. Additionally, the programming context is usually related to mathematics, making students not fluent in the topic even less confident in the possibility to solve the problem by using programming. Papert's Logo is a first programming language that provides concrete experience in programming through moving a turtle on the screen (Papert, 1980). Although the syntax in LOGO is simple, children still struggle with it, because every line of code has to be complete and precise (Flannery et al., 2013) within a context still related to mathematics. Papert argued that programming languages should have a "low floor" (easy to get started), "high ceiling" (opportunities to create increasingly complex projects over time) and "wide walls" (supporting many different types of projects so people with many different interests and learning styles can all become engaged) (Resnick et al., 2009). Novel block-based visual programming languages such as Scratch, Alice, Greenfoot, Kodu and others, satisfy the low-floor/high-ceiling/wide-walls triplet. Visual programming environments provide low barriers to artistic expression and civic engagement (Peppler and Kafai, 2007), and facilitate the development of the software in a fun and non-threatening context (Meerbaum-Salant et al., 2013).

Undergraduate students have chosen Scratch as the most effective block-based language for an introduction to programming for all educational levels (Xinogalos et al., 2017). So, it is appropriate to assume that Scratch is suitable for K-12 novice students as well. Instead of standard programming language syntax, Scratch commands use blocks shaped to fit together like puzzles; thus, preventing snapping together syntactically invalid commands. Therefore, "bugs" are always semantic and not syntactic as the result of a typing error or a misremembered detail of language syntax (Lewis, 2010). Considering all previously said we could conclude that Scratch is appropriate for teaching introductory programming at the K-12 level.

It supports diversity through many different kinds of projects (stories, games, animations, simulations) and personalisation by importing photos and music clips, recording voices, and creating graphics (Resnick et al., 2009), so the context is moved from mathematics to the "real" world, as expected by actual students. For all reasons mentioned above, high school students found block-based programming simpler than text-based, although they found block-based programming language to be more limited compared to text-based (Weintrop and Wilensky, 2015). It is important to mention that those were high-school students with previous programming experiences.

Reports show that we can successfully use Scratch for introductory programming at the undergraduate level. Between five visual programming environments; BlueJ; objectKarel; Scratch; Alice; and MIT App Inventor, all used for introducing programming through programming contexts like games and mobile apps, students reported Scratch as the most effective environment for an introduction to programming for all educational levels (Xinogalos et al., 2017).

2.3 Choosing the appropriate programming language

One of the main questions is, what programming language should novices learn first? We must take into account the students' age but also the way of learning.

All programming languages use programming constructs; they are used to control the order (flow) in which statements are executed (or not executed). The classic definitions were introduced by Böhm and Jacopini (Böhm and Jacopini, 1966) who demonstrated that programs built from the small set of regular and straightforward programs could represent flowchart programs of any size and complexity. Based on the result of their work, we define a structured program as any program describing the flow of execution control by using only the three basic programming constructs: sequences, selection and repetition (iteration or loop). All of the declared constructs are also program statements in their own right.

One of the described ways of programming builds on the stepwise refinement process – a method of problem decomposition common to all engineering disciplines and physical, chemical, and biological sciences (Jensen, 1981); therefore, we have concentrated our research on one of the basic programming concepts – the loop.

There is no doubt that K-12 and undergraduate programming novices are facing different difficulties.

According to Bruner (Bruner, 1966), learning occurs in three stages: enactive, iconic and symbolic. Manipulating concrete objects, like robots, could be considered as a tool for learning at the enactive stage that is appropriate even for the kindergarten age. Manipulating images of objects could be regarded as a tool for learning at the iconic stage. Scratch enables iconic learning but possibly enactive and symbolic learning. Manipulating representations of objects fit into the symbolic stage. Learning to program in text-based programming languages is almost entirely representational (Armoni et al., 2015).

As showed that visual illustrations help the children and that the pictures facilitate children's prose learning (Levin, 1981), by instructions visualisation, for example as puzzle pieces in Scratch, we might mitigate the problems in the understanding of complex programming concepts like the loop for K-12 students.

2.4 Loop construct

Loop construct is one of the core programming constructs. It is a complex and abstract concept, especially for novices, so it is essential to start with the simple examples, especially when it comes to K-12 students. Understanding the simple cases is a foundation for more complex cases; therefore, it is vital to master the simple before moving to more complex ones.

There are two basic kinds of loops in text-based programming languages: loops without conditional execution, like *repeat* (Logo), and *for* (Python and Logo); and loops

with conditional execution like *while* and *repeat-until*. Loops without conditional execution are easier to understand. *Repeat* and *for* loops repeats the execution as many times as stated.

In block-based programming languages, loops are different than in text-based programming languages, but the underlying concept is the same (Ben-Ari, 2011). Unbounded loop and fixed bounded loop fits in the loops without conditional execution, while conditional bounded loops and unbounded conditional loops fit into the loops with conditional execution. Table 1 shows the examples of loops without conditional execution in Scratch, Logo, and Python used in this research.

Table 1 Fixed bounded loops

Scratch	*LOGO*	*Python*
	repeat 10 […]	for *i* in range (10): …

As shown in Table 1 it is evident that Scratch syntax is the most intuitive and the only possible errors are semantic, not syntactic. That is not the case in Logo and Python where the Python syntax is a little bit more complex than Logo.

Students should first be familiar with these basic examples before moving on to more complex cases including conditionals and variables. That is one of the reasons for testing only this kind of loops in this study.

Researchers conducted most of the studies about loop concept understanding on the undergraduate level. There is lack of those studies for novices at the K-12 level.

Some studies conducted at the K-12 level compared understanding of some programming concepts in Scratch and Logo programming languages. One of those studies considered 6th graders in the summer camp. For understanding nested loops results showed no statistical differences between Scratch and Logo, but when it comes to conditionals, Scratch group achieved, statistically significant, better results (Lewis, 2010).

Another similar study, conducted in school settings among 7th -grade students, also compared students' success in Scratch and Logo programming languages. This study showed that Scratch group achieved statistically significant, better results for the nested loop (Mladenović et al., 2016b).

Mladenovic et al. (2017a) also reported that K-12 students' misconceptions about loop concept decreases when using Scratch as the first programming language compared to Python and Logo programming languages.

Weintrop conducted the study at high school in the school settings and compared students' success in block-based and text-based environments. Results showed that the majority of students perceived block-based programming easier than text-based programming. Students also performed significantly better for the block-based programming language for conditional logic, iterative logic (loops) and procedures. Although not significantly but they performed better on variables in the graphical condition (Weintrop and Wilensky, 2015).

Research conducted in the Israel high school also reported that students who learned Scratch before Java recognised the loop concept when they were introduced to repeated execution (Armoni et al., 2015).

There are several studies at the undergraduate level; therefore, these studies were reviewing the understanding of more complex examples of the loops, including variables and conditionals within loops. Soloway et al. found that only 38% of novices can correctly make the program for calculating the average of numbers (Soloway et al.,

1982). Results of another study conducted at the undergraduate level using the Java programming language showed that 60% of students were successful in tracing a while loop, and 32% of students were successful in predicting a return value of the procedure which consists of while loop (Lopez et al., 2008).

Another study showed that students understanding of the conditional and loop constructs are incomplete. Common novice mistake is ignoring the loop when the body has no control variables (Sekiya and Yamaguchi, 2013).

3 Methodology

This research is a part of the broader research. We examined results from different aspects. One aspect was to seek for students misconceptions for the loop concept. The next aspect of research refers to comparing student results in different programming languages based on students' problem-solving abilities and age. In this study, we have examined the second aspect.

3.1 Research questions

We have completed the research on understanding one of the basic concepts in programming – the loop. The research was conducted in three different programming languages using the same tests. The programming languages chosen were: Scratch, Logo, and Python using the turtle library.

Based on the theoretical background in the previous section, we have outlined two main research questions. The first research question is: Does the choice of programming language influence the performance of students when solving problems by programming? The second research question is: Will the programming success increase if we choose the programming language with concern for students' age?

Based on the research questions, we posed the following hypotheses:

H1: The scores achieved in post-test compared to the achievement of problem-solving pre-test will be higher when using Scratch than the scores achieved when using Logo or Python.

H2: The overall scores achieved using Scratch programming language will be higher than the scores achieved using Logo or Python programming languages.

H3: Novices (5th graders) will achieve higher scores in Scratch than those using Logo or Python for all questions.

3.2 Research context

Informatics is the elective course in the elementary school in Croatia, and programming is part of the curriculum for every grade (Ministry of Science Education and Sports of the Republic of Croatia, 2005). The curriculum follows the idea of the spiral curriculum (Bruner, 1960). The programming language is not strictly defined; instead, it specifies two possible approaches. One is the Logo-based approach and the second is a textual programming language approach so that the teacher can choose one of those. The Logo-based approach relies on Papert's theory of constructionism (Papert, 1980) and therefore, Scratch, Logo and Python using turtle library follow the same approach, which we found

appropriate for this research, as we were able to create the same tasks using different languages and compare students results. Differences between the languages used are the presentation of instructions – Scratch relies on block-based instructions while Logo and Python use textual instructions. Both approaches are in line with the curricula.

By using the Logo programming language, children can observe the outcome of the program by executing a program providing a graphics output. Students can create very complex figures using just a few instructions. The Logo was initially created to help understand geometry in math. The problem is that the instructions are still text-based and instructions are abbreviations from words in the English language. Table 2 shows basic instructions in the languages used.

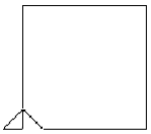
As can be seen from Table 2, instructions in Scratch are more precise and syntax-error free. Such programming language property is more significant as instructions become more difficult with each new programming environment.

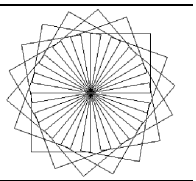
Table 2 Basic instructions

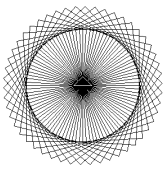
Instructions in Scratch	*Instructions in Logo (Python)*	*Abbreviation*	*Meaning*
	fd (100)	forward	Go forward for 100 steps
	rt (90)	right turn	Turn right for 90°
	lt (90)	left turn	Turn left for 90°

Tables 3 and 4 shows the research design for both groups.

Table 3 Logo and Python groups research design

Logo and Python groups (L-P groups)				
Week	*Topic*	*New instructions*	*New concepts*	*Program example*
1	Pre-test			
2	Drawing basic shapes	fd, bk, rt, lt, repeat Example of code: `repeat 4[fd 100 rt 90]`	Introducing algorithm term, basic algorithms: sequencing and repeating. Drawing basic shapes like triangle or square by moving a turtle using basic instructions, loops.	

3	Drawing shapes	Procedures, sub-procedures
		Example of code:
		`to cvijet`
		`repeat 18[kvadrat rt 360/18]`
		`end`
			Introducing to modularity by introducing procedures and sub-procedures for drawing more complex shapes.	

4	Drawing shapes	Procedures with parameters, variables
		Example of code:
		`to cvijet :n`
		`repeat :n[kvadrat rt 360/:n]`
		`end`
			Introducing to variables as parameters for procedures.	

5	Post-test			

Table 4 Scratch group research design

Scratch groups (S-groups)				
Week	*Topic*	*New instructions*	*New concepts*	*Program example*
1		Pre-test		
2	Aquarium program	forward, left, right, move, wait, next costume, forever Example of code: 	Introducing algorithm term, basic algorithms: sequencing, branching and repeating. Making the aquarium simulation by moving sprites using basic instructions, concurrency and loop concepts	
3	Chasing ghosts game	If, variables Example of code: 	Introducing conditionals and variables by programming "Chasing ghosts" game.	
4	Simple ricochet game	Broadcast message, forever if Example of code: 	Introducing loops with conditionals and broadcasting by programming simple ricochet game.	
5		Post-test		

As Tables 3 and 4 show, at the same amount of time, it is possible to process more concepts in block-based programming language than in text-based programming language. For example, the conditionals and conditional loops are not introduced to L-P groups at all for its complexity and lack of time for students to properly master the mentioned concepts. Those concepts are used intuitively by programming games in the S group, so it is much easier to introduce them to students. It is important to mention that every lecture included the loop concept. That is one of the reasons to investigate the understanding of the loop concept in this study.

3.3 Participants

Researchers conducted the research in school settings with 312 students in four elementary schools in Split, Croatia from 5th to 8th grade (11 to 14 years old) during the 2015/2016 school year. All students attended an elective course in informatics; therefore, conducted tests followed the official curriculum. Sampling was non-probability, purposive (Louis et al., 2011) since the goal was to examine students enrolled in the elective informatics course. Table 5 shows the research population.

Table 5 Population details

School	*Class*	*Language*	*Students*	*Boys*	*Girls*	*Previous programming experience*
S1	5	Scratch	30	14	16	No
S1	6	Python	17	7	10	Yes(Python)
S2	5	Logo	48	22	26	No
S2	6	Logo	54	31	23	Yes(Logo)
S2	7	Logo	35	18	17	Yes(Logo)
S2	8	Logo	48	30	18	Yes(Logo)
S3	5	Python	51	20	31	No
S4	6	Scratch	16	11	5	No
S4	7	Scratch	13	6	7	No
	Total:		312	159	153	

As programming is part of the informatics curriculum from 5th to 6th grade, some students had previous programming experience in a programming language studied in school. Informatics is an elective course, so students can drop out or join the course after the end of each school year, which is why one 6th and 7th -grade class did not have previous programming experience; thus, all Scratch groups were novices. When using Logo approach, learners studied the loop concept in 5th grade.

3.4 Procedures and instruments

Previously, researchers observed multiple concepts and found differences in the understanding of loops. Students that used Scratch achieved higher results in comparison to students using Logo (Mladenović et al., 2016b). That is why we decided to focus on the loop understanding in this study. With the Logo approach, the first programming concept considered is sequencing, for which we created an assignment. The next programming concept to learn is the loop, which is an integral part of the curricula

through all grades. In Logo and Python, students were introduced to the loop concept conventionally, by drawing different shapes for approximately four weeks, two hours per week. In Scratch, students programmed games designed to teach the loop concept for three weeks, two hours per week.

For the analysis, we have used a quantitative approach. The research tools included two tests to obtain quantitative results on the learning of concepts.

3.4.1 Pre-test

We have conducted the pre-test before exposure to programming lectures. The test was designed to check the problem-solving abilities of students. As mentioned before, the problem-solving ability is indispensable for programming, but also by programming we can practice the problem-solving. The test was composed of eight questions and did not require advanced math skills but the problem-solving abilities with an emphasis on choosing the right algorithm. The test was made more simple than it should be so that the weaker students would not feel uncomfortable and unmotivated. Mladenović et al. previously conducted the same problem-solving test, which showed that problem-solving is not crucial when it comes to success in Python programming (Mladenović et al., 2016a). The scale had a high level of internal consistency, as determined by Cronbach's alpha of 0.709.

3.4.2 Post-tests

Post-tests were conducted a week after programming lectures. Tests were identical with the only difference being the programming language used. The first task surveyed sequencing while the other four tasks examined loop concept understanding. The question for every task was:

How many steps will turtle or sprite make after the execution of the following code? (Circle the correct answer)

The question was identical for all the languages. Table 6 shows the answers and the respective code.

The questions are ordered from hardest to easiest as follows: Q4, Q5, Q3, Q2, Q1, so when evaluating, every answer is evaluated by the weight factor in the weight order, so the total number of points was 14. A Cronbach's alpha of 0.768 approves reliability of the test.

Table 6 Codes and answers in Scratch, Logo and Python

Questions and answers	*Scratch*	*Logo*	*Python*	*Answers*
Q1		fd 100 lt 90 fd 50 fd 50 lt 90	fd (100) lt (90) fd (50) fd (50) lt (90)	a) 100 b) 150 c) 180 d) 200 e) 380

Table 6 Codes and answers in Scratch, Logo and Python (continued)

Questions and answers	*Scratch*	*Logo*	*Python*	*Answers*
Q2		fd 10 repeat 10[fd 10]	fd(10) for i in range (10): fd(10)	a) 10 b) 20 c) 30 d) 100 e) 110
Q3		repeat 10[fd 10 rt 10 fd 10]	for i in range (10): fd(10) rt(10) fd(10)	a) 20 b) 30 c) 40 d) 200 e) 300
Q4		repeat 10[fd 1 repeat 10[fd 1]]	for i in range (10): fd(1) for i in range (10): fd(1)	a) 10 b) 20 c) 100 d) 110 e) 1000
Q5		fd (1) repeat 10[fd 1] repeat 10[fd 1]	fd(1) for i in range (10): fd(1) for i in range (10): fd(1)	a) 20 b) 21 c) 23 d) 110 e) 103

4 Results and discussion

In this section, we describe the study results. Our research questions are composed of four hypotheses: H1, H2, and H3. The next sections present results for these hypotheses.

4.1 Normality of data

Kolmogorov-Smirnov test results showed that pre-test results are not normally distributed ($n = 312$, $p = 0.000$).

We conducted the post-tests for three groups: Python, LOGO and Scratch. The Kolmogorov-Smirnov test results showed that post-test results for Python ($n = 68$, $p = 0.000$), LOGO ($n = 185$, $p = 0.000$) and Scratch ($n = 59$, $p = 0.000$) are also not normally distributed.

Non-parametric tests are used for further testing as the pre-test, and post-test data sets are not normally distributed.

4.2 Comparing students' post-test results based on pre-tests

To prove H1 paired samples t-tests were used to compare pre-test results and post-test results by the programming language used. Since pre-test and post-tests are not equivalent, we matched post-test student scores based on pre-test success and age. We used precise matching, which means that we compared students with the same pre-test points and the same age. Among 312 students, we extracted 32–37 pairs of students from 5^{th} and 6^{th} grades. Wilcoxon signed-rank test is used to compare students' post-test results based on pre-tests. Table 7 shows the results of the Wilcoxon signed-rank test.

Table 7 Results of the Wilcoxon signed-rank test

Pairs	*N*	*z*	*df*	*p*
Logo - Scratch	37	–2.97	36	0.003*
Python - Scratch	33	–4.116	32	0.000*
Python - Logo	32	–1.705	31	0.088

Results showed that Scratch scores are statistically significantly higher than those for Logo (Z = –2.97, p = 0.003) the same applies to Python (Z = –4.116, p = 0.00) as presented in Table 6. When comparing Logo and Python, the results (Z = –1.705, p = 0.088) did not show statistically significant differences. Table 8 shows the descriptive statistics. All the results confirmed a positive correlation between pre-tests and post-tests, as expected.

Table 8 Descriptive statistics by languages

Pairs	*Language*	*N*	*Mean*	*SD*
Scratch-Logo	Scratch	37	10.97	4.42
	Logo	37	8.78	3.44
Scratch-Python	Scratch	33	11.12	4.49
	Python	33	7.45	3.79
Logo-Python	Logo	32	8.59	3.43
	Python	32	7.22	3.60

It is evident that students of the same age and with the same problem-solving abilities achieve different results when using different programming languages. Results showed that students achieved higher results in Scratch in comparison to Logo or Python; therefore, there is less chance that student will give up on programming. Hence, it is a significant finding, which can favour using Scratch for age 11 to 13. According to all the above, we can accept H1 and conclude that not only are the scores achieved in Scratch higher than the results achieved in the pre-test but also, the scores in Logo and Python are worse than the pre-test.

4.3 Compare by language

To prove H2 we used Kruskal-Wallis test to compare results between groups based on the programming language used. Results showed statistically significant differences ($\chi^2(2) = 43.323$, $p = 0.000$), so for further analysis, the Mann-Whitney U test was used to determine where the differences are. Table 9 shows the results of the Mann-Whitney U test between programming languages.

Table 9 Results of the Mann-Whitney U test between programming languages

		Pre-test	*Q1*	*Q2*	*Q3*	*Q4*	*Q5*	*Total*
Scratch vs. Logo	U	5201.5	3698	3694	3726	4545	4032	2727.5
	p	0.586	0.00*	0.00*	0.00*	0.02*	0.00*	0.00*
Scratch vs. Python	U	1695	1225	1309	1345	1138	1409	816
	p	0.131	0.00*	0.00*	0.00*	0.00*	0.00*	0.00*
Logo vs. Python	U	5698	5869	6137	6213	4620	6061	5402
	p	0.25	0.346	0.732	0. 860	0.000*	0.593	0.08

As shown in Table 9, Mann-Whitney U showed statistically significant differences for Scratch groups compared with Logo and Python for all questions. When comparing Logo and Python, a statistically significant difference is shown for Q4. To determine where the differences are, we looked for means, as shown in Table 10.

Table 10 Means for programming languages by questions

Question (max. Point number)	*Q1 (1)*	*Q2 (2)*	*Q3 (3)*	*Q4 (5)*	*Q5 (4)*	*Total (15)*
Total	0.564	1.17	1.40	1.29	2.74	4.72
Scratch	0.83	1.70	2.14	2.46	3.67	10.78
Logo	0.51	1.04	1.18	1.62	2.62	6.98
Python	0.44	1.00	1.15	0.29	2.47	5.35

From Table 8 we can conclude that students achieve higher results in Scratch for every question when compared to Logo and Python. Results from Logo and Python showed statistically significant differences for Q4 where students achieved higher scores in Logo than Python but also in the overall test. As we can see, the main observed difference is in Q4, which is the most challenging question containing a nested loop.

Based on the results we can accept H2 and conclude that students achieved higher scores in Scratch than in Logo or Python.

4.4 Comparison for 5th grade (novices)

Students from 5th grade (11 years old) are all novices in programming, so we found it interesting to compare them by language for all questions. To prove H3, we use the Mann-Whitney U test. Table 11 shows the results of the Mann-Whitney U tests.

Table 11 Results of the Mann-Whitney U test between programming languages for 5th-grade students

		Q1	*Q2*	*Q3*	*Q4*	*Q5*	*Total*
Scratch vs. Logo	U	543	504	504	513	552	362.5
	p	0.031*	0.008*	0.010*	0.005*	0.20*	0.00*
Scratch vs. Python	U	448.5	528	505.5	448.5	466.5	302
	p	0.00*	0.006*	0.003*	0.00*	0.00*	0.00*
Logo vs. Python	U	1018.5	1212	1176	1069.5	1032	989.5
	p	0.094	0.923	0.676	0.022*	0.115	0.094

Results showed statistically significant differences when comparing Scratch with Logo and Python for all questions, as shown in Table 11. The only statistically significant difference between Logo and Python was for Q4 and total results. Table 12 shows the means for all questions.

Table 12 Means for programming languages by questions for 5th-grade students

Question (max. Point number)	*n*	*Q1 (1)*	*Q2 (2)*	*Q3 (3)*	*Q4 (5)*	*Q5 (4)*	*Total (15)*
Total	129	0.51	1.13	1.16	0.81	2.64	6.26
Scratch	30	0.77	1.60	1.90	2.17	3.60	10.03
Logo	48	0.52	1.00	1.00	0.72	2.67	5.91
Python	51	0.35	0.98	0.88	0.098	2.03	4.35

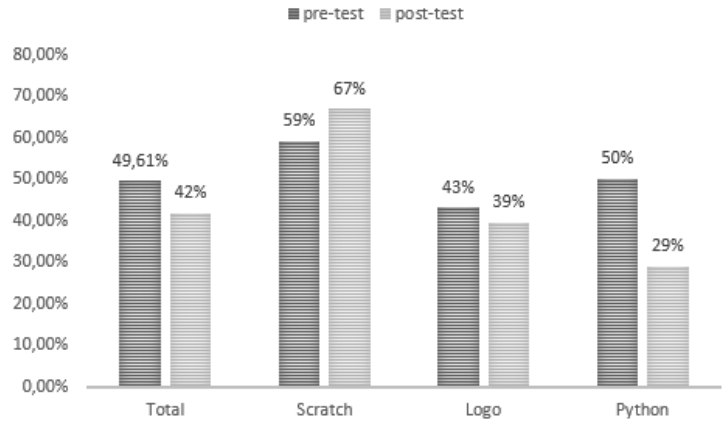
From Table 11 it is evident that students achieved higher scores in Scratch than in Logo and Python for all questions, so we can accept H3 and conclude that students achieved higher scores in Scratch than in Logo or Python for each examined concept.

From Table 12 we can also see that there are no statistical differences for total results but just for Q4. We can conclude that higher scores achieved in Scratch relate to Q4 since that was the only question with the statistically significant difference. Results from 5th graders have probably a more important value since they are all novices. It is important to note that students did not prepare in advance for the test; they were studying lessons from the curriculum that do not include nested loops so we can attribute the observed difference to differences in the syntax of languages used. Python uses indentations as parts of loop syntax, so students at this age have problems with nested loops due to double indentation usage for the second loop, which seems like they do not recognise it as another loop. On the other hand, in Logo, parentheses are used as a loop limiter, which is less confusing.

Pre-test mean for 5th graders (n = 129) is 49.61. Moreover, when comparing scores of the pre- and post-test, the only positive difference is observed when using Scratch while in both Logo, and Python students performed not as well as while solving the pre-test, as shown in Figure 1.

Based on the results we can also note that the efficiency in programming depends on the chosen programming language, which probably bases on the students' stage of development.

Figure 1 Scores of the pre-tests and post-tests for 5th graders



4.5 Limitations of the study

Another limitation of the test design is the curriculum. The test aims to explore the understanding of the basic loop use cases for several reasons. Although we could have a better insight into the loop concept understanding by testing the iterative loop involving variables this was not feasible for various reasons. According to the curriculum, teaching topics are determined, so the test design depends on it. We intended to explore the understanding of the loop concept with as many students as possible leading to the test design including learning topics for grades from 5th to 7th. As variables, conditionals and for loops, based on curriculum, are defined as teaching topics in 6th grade it was not possible to include loops with counters nor conditionals. As 5th graders are all programming novices, they are the most interesting group to examine, so the test was designed to include that population.

Furthermore, most studies about understanding programming concepts were conducted at the undergraduate level. Consequently, all K-12 students could be considered being novices. According to Robins et al. "It is clear that novice programmers face a challenging task. Learning to program involves acquiring complex new knowledge and related strategies and practical skills. Hence first-course material should be simple, and this should be expanded on systematically as the students gain experience" (Robins et al., 2003). Therefore, students should first be familiar with the base cases of those complex concepts. As the goal is to foster deep learning in students, and many students make little progress in a first programming course (Robins et al., 2003), particularly at the K-12 level, by choosing the proper programming environment, programming context, and level of the complexity could set the basis for in-depth learning of the programming concepts later.

4.6 Implications for teaching and future research

In this research, we tested the understanding of one of the basic programming concepts – loop in three programming languages, Scratch, Logo and Python – among 5th to 8th - grade elementary school students. Students at that age need concrete experience to understand abstract concepts, which Scratch as a visual programming language offers. At that age, they are mostly not ready for text-based programming languages so the consequence of using text-based languages too early can lead to burnout and increase drop-out from programming and CS as well. Results of this research suggest that students with the same problem-solving abilities achieved higher test scores when using Scratch than when using Logo or Python. There were no differences between Logo and Python scores.

The implications of the key findings in this research could provide significant benefits for students and teachers as well. As concepts used in programming are abstract, the teacher should consider the ways to provide concrete experience to facilitate the understanding of underlying abstract concepts. Results of this research, showed that programming novices are achieving better results when using block-based programming language compared to students using textual-programming languages thus confirming that use of the block-based programming language is providing concrete to abstract experience (Dann and Cooper, 2009) of the underlying abstract concepts. Furthermore, by using visual programming language, known novices syntax problems in text-based programming languages are also minimised.

Therefore, teachers could make abstract programming concepts more concrete by using visual programming language, like Scratch and shifting the programming context from solving math problems towards the programming of computer games, storytelling and others.

Question is how to transfer from visual programming environments to the "real" text-based programming languages? The question is motivating our future research.

Based on the results of the research presented in this paper but also in Mladenović et al. (2017a) the next phase would focus on the transfer from block-based to a text-based programming language.

5 Conclusion

The research focused on the use of the three programming languages – Scratch, Logo and Python – to teach one of the basic programming concepts, the loop (repeated execution).

The loop is an abstract concept that is difficult to understand and embrace, especially when using the text-based programming languages like Logo or Python. Hence, we decided to use Scratch, a block-based visual language, to be able to compare the results achieved by students while using three selected languages. We conducted the research in a school setting among 312 students from 5^{th} to 8^{th} grade in four elementary schools. Tests for all languages were equivalent, differing only in the used programming language. Before the beginning of the programming lectures, students were tested to determine their problem-solving ability as one of the abilities needed for programming.

We found that students achieved higher scores when using the Scratch programming language, thus confirming our assumptions. Differences in students' achievement while using Logo and Python existed when solving the most complex task with a nested loop included, where the scores are higher in Logo than in Python. Those differences are probably related to syntax differences in respective programming languages. Python uses indentations to indicate the "for" statement while Logo uses brackets for the same purpose. Revealed problems are non-existent in Scratch since the language syntax is block-based. Thus, children do not have to struggle with the syntax, but instead, they can focus on problem-solving. The last, but maybe the most important finding, is related to the achievement while using these programming languages based on the problem-solving pre-test. Students attained higher scores when using Scratch compared to the pre-test, while students programming in Logo and Python attained inferior results compared to the pre-test. Results confirm the assumption that abstract concepts are easier to understand when providing the concrete experience by using the block-based programming language like Scratch. Those findings described above can lead us to the conclusion that if we neglect the students' age while choosing the programming language, instead of using programming as a problem-solving ability booster tool, we can create a "problem-making" environment resulting in negative bias on students' problem-solving ability and self-confidence.

References

Armoni, M., Meerbaum-Salant, O. and Ben-Ari, M. (2015) 'From Scratch to "real" programming', *ACM Transactions on Computing Education*, Vol. 14, No. 4, pp.25:1–25:15. Available online at: http://dl.acm.org/citation.cfm?doid=2698235.2677087 (accessed March 7, 2017).

Ben-Ari, M. (Moti) (2011) 'Loop Constructs in scratch', *ACM Inroads*, Vol. 2, No. 1, p.27. Available online at: http://dl.acm.org/citation.cfm?doid=1929887.1929900 (accessed August 30, 2017).

Ben-Bassat Levy, R., Ben-Ari, M. and Uronen, P.A. (2003) 'The Jeliot 2000 program animation system', *Computers & Education*, Vol. 40, No. 1, pp.1–15. Available online at: http://linkinghub.elsevier.com/retrieve/pii/S0360131502000763 (accessed March 7, 2017).

Biggers, M., Brauer, A. and Yilmaz, T. (2008) 'Student Perceptions of Computer Science: A Retention Study Comparing Graduating Seniors with Cs Leavers', *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '08. New York, NY, USA: ACM, pp. 402–406.

Böhm, C. and Jacopini, G. (1966) 'Flow diagrams, turing machines and languages with only two formation rules', *Communications of the ACM*, Vol. 9, No. 5, pp.366–371. Available online at: http://dl.acm.org/citation.cfm?id=365646.

Bruner, J.S. (1960) 'The Process of Education', *The Process of Education*. pp. 12–13.

Bruner, J.S. (1966) *Toward a theory of instruction*, Harvard University Press.

Cassel, L. (Boots) et al. (2007) 'The Current Crisis in Computing: What Are the Real Issues?' *SIGCSE Bull.*, Vol. 39, No. 1, pp.329–330.

Cohen, L., Manion, L. and Morrison, K. (2011) *Research Methods in Education*, Oxford, UK: Routledge.

Dann, W. and Cooper, S. (2009) 'Alice 3: Concrete to Abstract', *Communications of the ACM*, Vol. 52, No. 8, p.27.

Dann, W. et al. (2012) 'Mediated Transfer : Alice 3 to Java', *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pp.141–146.

Denning, P.J. (2009) 'The Profession of IT: Beyond Computational Thinking', *Commun. ACM*, Vol. 52, No. 6, pp.28–30.

Dorling, M. and White, D. (2015) 'Scratch : A Way to Logo and Python', *SIGCSE '15: Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pp.191–196.

Fincher, S. and Utting, I. (2010) 'Machines for Thinking', *Trans. Comput. Educ.*, Vol. 10, No. 4, p.13:1--13:7.

Flannery, L.P. et al. (2013) 'Designing ScratchJr: Support for Early Childhood Learning Through Computer Programming', *Proceedings of the 12th International Conference on Interaction Design and Children (IDC '13)*, pp.1–10.

Gomes, A. and Mendes, A.J.N. (2007) 'Learning to program-difficulties and solutions', *International Conference on Engineering Education*, pp.1–5.

Honey, M. (2014) *STEM Integration in K-12 Education: Status, Prospects, and an Agenda for Research*, The National Academies Press.

Jensen, R.W. (1981) 'Tutorial Series 6 Structured Programming', *Computer*, Vol. 14, No. 3, pp.31–48.

Kinnunen, P. and Malmi, L. (2006) 'Why Students Drop out CS1 Course?' *Proceedings of the Second International Workshop on Computing Education Research*. ICER '06. New York, NY, USA: ACM, pp. 97–108.

Kramer, J. (2007) 'Is Abstraction the Key to Computing?' *Commun. ACM*, Vol. 50, No. 4, pp.36–42.

Lahtinen, E., Ala-Mutka, K. and Järvinen, H.-M. (2005) 'A study of the difficulties of novice programmers', *ACM SIGCSE Bulletin*, Vol. 37, No. 3, p.14. Available online at: http://portal.acm.org/citation.cfm?doid=1067445.1067453 (accessed May 24, 2017).

Levin, J.R. (1981) 'On Functions of Pictures in Prose', *Neuropsychological and Cognitive Processes in Reading*. Elsevier, pp. 203–228. Available online at: http://linkinghub.elsevier.com/retrieve/pii/B9780121850302500135 (accessed September 7, 2017).

Lewis, C.M. (2010) 'How programming environment shapes perception, learning and goals', *Proceedings of the 41st {ACM} technical symposium on Computer science education - {SIGCSE} {\textquotesingle}10*. Association for Computing Machinery ({ACM}).

Lopez, M. et al. (2008) 'Relationships between reading, tracing and writing skills in introductory programming', *Proceeding of the fourth international workshop on Computing education research - ICER '08*. New York, New York, USA: ACM Press, pp. 101–112. Available online at: http://portal.acm.org/citation.cfm?doid=1404520.1404531 (accessed August 30, 2017).

Malan, D.J. et al. (2007) 'Scratch for budding computer scientists', *Proceedings of the 38th SIGCSE technical symposium on Computer science education - SIGCSE '07*. New York, USA: ACM Press, p. 223. Available online at: http://portal.acm.org/citation.cfm?doid=1227310. 1227388 (accessed March 6, 2017).

Maloney, J.H., Peppler, K., Kafai, Y., Resnick, M. and Rusk, N. (2008) 'Programming by choice', *ACM SIGCSE Bulletin*, Vol. 40, No. 1, p.367. Available online at: http://portal.acm.org/citation.cfm?doid=1352322.1352260 (accessed March 6, 2017).

Maloney, J.H., Peppler, K., Kafai, Y., Resnick, M., Rusk, N., et al. (2008) 'Programming by choice', *ACM SIGCSE Bulletin*, Vol. 40, No. 1, p.367. Available online at: http://portal.acm.org/citation.cfm?doid=1352322.1352260 (accessed March 31, 2017).

Maltese, A. V. and Tai, R.H. (2010) 'Eyeballs in the fridge: Sources of early interest in science', *International Journal of Science Education*, 32(December 2014), pp.669–685.

McCracken, M. et al. (2001) 'A multi-national, multi-institutional study of assessment of programming skills of first-year CS students', *ACM SIGCSE Bulletin*, Vol. 33, No. 4, p.125. Available online at: http://portal.acm.org/citation.cfm?doid=572139.572181 (accessed May 24, 2017).

McKenna, P. (2004) 'Gender and black boxes in the programming curriculum', *Journal on Educational Resources in Computing*, Vol. 4, No. 1, p.6--es.

Meerbaum-Salant, O., Armoni, M. and Ben-Ari, M. (Moti) (2013) 'Learning computer science concepts with Scratch', *Computer Science Education*, Vol. 23, No. 3, pp.239–264.

Ministry of science education and Sports of the Republic of Croatia (2005) *The curriculum for primary school*, Zagreb.

Mladenović, M., Boljat, I. and Žanko, Ž. (2017a) 'Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level', *Education and Information Technologies*, pp.1–18. Available online at: http://link.springer.com/10.1007/s10639-017-9673-3.

Mladenović, M., Krpan, D. and Mladenović, S. (2016) 'Introducing programming to elementary students novices by using game development in Python and Scratch', *EDULEARN16 Proceedings*. 8th International Conference on Education and New Learning Technologies. IATED, pp. 1622–1629. Available online at: http://dx.doi.org/10.21125/edulearn.2016.1323.

Mladenović, M., Krpan, D. and Mladenović, S. (2017b) 'Learning programming from Scratch', *International Conference on New Horizons in Education INTE*.

Mladenović, M., Rosić, M. and Mladenović, S. (2016) 'Comparing Elementary Students ' Programming Success Based on Programming Environment', *International Journal of Modern Education and Computer Science*, 8(August), pp.1–10.

Moser, R. (1997) 'A Fantasy Adventure Game As a Learning Environment: Why Learning to Program is So Difficult and What Can Be Done About It', *SIGCSE Bull.*, Vol. 29, No. 3, pp.114–116.

Papert, S. (1980) *Mindstorms: Children, Computers, and Powerful Ideas*, New York, NY, USA: Basic Books, Inc.

Peppler, K.A. and Kafai, Y.B. (2007) 'From SuperGoo to Scratch: exploring creative digital media production in informal learning', *Learning, Media and Technology*, Vol. 32, No. 2, pp.149–166. Available online at: http://www.tandfonline.com/doi/abs/10.1080/17439880701343337.

Repenning, A., Webb, D. and Ioannidou, A. (2010) 'Scalable game design and the development of a checklist for getting computational thinking into public schools', *Proceedings of the 41st ACM technical symposium on Computer science education*. pp. 265–269.

Resnick, M. et al. (2009) 'Scratch: Programming for All', *Commun. ACM*, Vol. 52, No. 11, pp.60–67.

Robins, A., Rountree, J. and Rountree, N. (2003) 'Learning and teaching programming: A review and discussion', *Computer science education*, Vol. 13, No. 2, pp.137–172.

Sanders, K. and Thomas, L. (2007) 'Checklists for Grading Object-oriented CS1 Programs: Concepts and Misconceptions', *SIGCSE Bull.*, Vol. 39, No. 3, pp.166–170.

Sekiya, T. and Yamaguchi, K. (2013) 'Tracing quiz set to identify novices' programming misconceptions', *Proceedings of the 13th Koli Calling International Conference on Computing Education Research - Koli Calling '13*. New York, New York, USA: ACM Press, pp. 87–95. Available online at: http://dl.acm.org/citation.cfm?doid=2526968.2526978 (accessed May 24, 2017).

Smith, D.C., Cypher, A. and Tesler, L. (2000) 'Programming by example: novice programming comes of age', *Communications of the ACM*, Vol. 43, No. 3, pp.75–81. Available online at: http://portal.acm.org/citation.cfm?doid=330534.330544 (accessed June 4, 2018).

Soloway, E. et al. (1982) 'What do novices know about programming', *Directions in human-computer interaction*, pp.87–122.

Tai, R.H. (2006) 'Career choice. Planning early for careers in science', *Science*, Vol. 312(5777), pp.1143–1144.

Turkle, S. and Papert, S. (1992) 'Epistemological Pluralism and the Revaluation of the Concrete', *Journal of Mathematical Behavior*, Vol. 11, No. 1, pp.3–33.

Violino, B. (2009) 'Time to Reboot', *Commun. ACM*, Vol. 52, No. 4, p.19.

Webb, M. et al. (2016) 'Computer science in K-12 school curricula of the 2lst century: Why, what and when?' *Education and Information Technologies*, pp.1–24.

Weintrop, D. (2015) 'Minding the Gap Between Blocks-Based and Text-Based Programming', *Proceedings of the 46th ACM Technical Symposium on Computer Science Education - SIGCSE '15*. New York, New York, USA: ACM Press, pp. 720–720. Available online at: http://dl.acm.org/citation.cfm?doid=2676723.2693622 (accessed August 28, 2017).

Weintrop, D. and Wilensky, U. (2015) 'To block or not to block, that is the question', *Proceedings of the 14th International Conference on Interaction Design and Children - IDC '15*. New York, USA: ACM Press, pp. 199–208. Available online at: http://dl.acm.org/citation.cfm?doid=2771839.2771860 (accessed August 28, 2017).

White, G.L. and Sivitanides, M.P. (2009) 'A Theory of the Relationships between Cognitive Requirements of Computer Programming Languages and Programmers' Cognitive Characteristics', *Journal of Information Systems Education*, Vol. 13, No. 1, pp.59–66.

Xinogalos, S., Satratzemi, M. and Malliarakis, C. (2017) 'Microworlds, games, animations, mobile apps, puzzle editors and more: What is important for an introductory programming environment?' *Education and Information Technologies*, Vol. 22, No. 1, pp.145–176. Available online at: http://link.springer.com/10.1007/s10639-015-9433-1 (accessed June 4, 2018).

Yardi, S. and Bruckman, A. (2007) 'What is Computing?: Bridging the Gap Between Teenagers' Perceptions and Graduate Students' Experiences', *Proceedings of the Third International Workshop on Computing Education Research*. ICER '07. New York, NY, USA: ACM, pp. 39–50.